%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%function main()主函数 %%

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%清除所有内存、窗口和命令行内容

clc

clear all

close all

%% 程序总执行时间: T 计时开始

tic

%% 输入数据参数设置

inputdata\_options.classes\_num=3;%故障种类，若更改则Add\_Original\_Data函数中的标签也需要改

inputdata\_options.sample\_total=10000;%样本点总数,可任意更改，但是必须是sample\_num的整数倍

inputdata\_options.sample\_num=14;%变量个数，可任意更改

inputdata\_options.n\_columns=1000;%数据维度，可任意更改

%% 加载原始数据

fprintf('正在加载数据......');

[Input\_Data]=Add\_Original\_Data(inputdata\_options);

%% 设置AE模型参数

hidden.num1=13;%第一隐层神经元个数

hidden.num2=22;%第二隐层神经元个数

hidden.num3=40;%第三隐层神经元个数

hidden.net\_trainParam\_epochs=500;%迭代次数

hidden.learn\_rare=0.01;%AE学习率（0-1之间，越大越容易过拟合，越小收敛速度越慢）

%% 选择是离线建模还是在线预测

n=input('请选择1（离线建模）或者2（在线预测）：');

switch n

case 1

%% 调用DNN子程序

dnn(Input\_Data,inputdata\_options,hidden);

%% 概率值作为分类依据的总测试精度

load probabilitypred

pred1\_2= pred\_test\_original;

pred2\_2= pred\_test\_slope;

pred3\_2= pred\_test\_curvature;

%概率值归一化

original=exp(probability\_original);

probability\_original\_1=zeros(size(original,1),size(original,2));

for i=1:size(original,2)

for j=1:size(original,1)

probability\_original\_1(j,i)=original(j,i)/sum(original(:,i));

end

end

slope=exp(probability\_slope);

probability\_slope\_1=zeros(size(slope,1),size(slope,2));

for i=1:size(slope,2)

for j=1:size(slope,1)

probability\_slope\_1(j,i)=slope(j,i)/sum(slope(:,i));

end

end

curvature=exp(probability\_curvature);

probability\_curvature\_1=zeros(size(curvature,1),size(curvature,2));

for i=1:size(curvature,2)

for j=1:size(curvature,1)

probability\_curvature\_1(j,i)=curvature(j,i)/sum(curvature(:,i));

end

end

% n=input('请选择1（未归一化）或者2（归一化）：');

n=1;

switch n

case 1

% probability\_original=probability\_original;

% probability\_slope=probability\_slope;

% probability\_curvature=probability\_curvature;

case 2

probability\_original=probability\_original\_1;

probability\_slope=probability\_slope\_1;

probability\_curvature=probability\_curvature\_1;

otherwise

disp('输入错误！');

end

%找出概率最大的值作为分类结果

[predp1,xulie1]=max(probability\_original);

[predp2,xulie2]=max(probability\_slope);

[predp3,xulie3]=max(probability\_curvature\_1);

predpp=[predp1;predp2;predp3];

predppp=max(predpp);

fenlei=zeros(1,length(testLabels));

for i=1:length(testLabels)

if predppp(i)==predp1(i)

fenlei(i)=xulie1(i);

elseif predppp(i)==predp2(i)

fenlei(i)=xulie2(i);

elseif predppp(i)==predp3(i)

fenlei(i)=xulie3(i);

end

end

figure

plot(fenlei,'r\*');

% 画出测试数据的实际类别

hold on

plot(testLabels,'b+');

title('概率融合结果');

h=legend('预测','实际','location','northwest');

set(h,'Box','off');

set(h,'Fontsize',12);

%概率值作为分类依据总测试精度

accProb = mean(testLabels(:) == fenlei(:));

%% 决策级分类融合策略

pred2=zeros(1,length(testLabels));

for i=1:length(testLabels)

if pred1\_2(i)==1&&pred2\_2(i)==1&&pred3\_2(i)==1

pred2(i)=1;

elseif pred1\_2(i)==2&&pred2\_2(i)==2&&pred3\_2(i)==2

pred2(i)=2;

elseif pred1\_2(i)==3&&pred2\_2(i)==3&&pred3\_2(i)==3

pred2(i)=3;

else

if (pred1\_2(i)==1&&pred2\_2(i)==1)||(pred1\_2(i)==1&&pred3\_2(i)==1)||...

(pred2\_2(i)==1&&pred3\_2(i)==1)

pred2(i)=1;

elseif (pred1\_2(i)==2&&pred2\_2(i)==2)||(pred1\_2(i)==2&&pred3\_2(i)==2)||...

(pred2\_2(i)==2&&pred3\_2(i)==2)

pred2(i)=2;

elseif (pred1\_2(i)==3&&pred2\_2(i)==3)||(pred1\_2(i)==3&&pred3\_2(i)==3)||...

(pred2\_2(i)==3&&pred3\_2(i)==3)

pred2(i)=3;

else

if i>9

s=zeros(1,10);

sum1=0;

sum2=0;

sum3=0;

for j=1:10

s(j)=pred2(i-j);

if s(j)==1

sum1=sum1+1;

elseif s(j)==2

sum2=sum2+1;

else

sum3=sum3+1;

end

end

A=[sum1,sum2,sum3];

maxRes=max(A);

if maxRes==1

pred2(i)=1;

elseif maxRes==2

pred2(i)=2;

else

pred2(i)=3;

end

else

pred2(i)=1;

end

end

end

end

figure

plot(pred2,'r\*');

% 画出测试数据的实际类别

hold on

testLabels=Input\_Data.testLabels;

plot(testLabels,'b+');

title('决策级融合结果')

h=legend('预测','实际','location','northwest');

set(h,'Box','off');

set(h,'Fontsize',12);

% 决策级融合总测试精度

acc = mean(testLabels(:) == pred2(:));

%% 程序执行总时间

T=toc;

%% 保存数据到xls表格中去

fprintf('正在保存数据......\n');

sgc\_exist = exist('DNNresult.xls', 'file');

if sgc\_exist==0

c=2;

save c c

else

load c

c=c+1;

save c c

end

d = {'数据纬度','变量数', '隐层1神经元数','隐层2神经元数','隐层3神经元数','AE学习率','迭代次数',...

'训练原始分类率','测试原始分类率','训练斜率分类率',...

'测试斜率分类率','训练曲率分类率','测试曲率分类率','以概率的总分类率','以决策的总分类率',...

'总运行时间(分钟)'};

xlswrite('DNNresult.xls', d, 'result', 'A1');

load classesResult;

xlswrite('DNNresult.xls', inputdata\_options.n\_columns,'result', strcat('A',num2str(c)))

xlswrite('DNNresult.xls', inputdata\_options.sample\_num, 'result', strcat('B',num2str(c)))

xlswrite('DNNresult.xls', hidden.num1, 'result', strcat('C',num2str(c)))

xlswrite('DNNresult.xls', hidden.num2, 'result', strcat('D',num2str(c)))

xlswrite('DNNresult.xls', hidden.num3, 'result', strcat('E',num2str(c)))

xlswrite('DNNresult.xls', hidden.learn\_rare, 'result', strcat('F',num2str(c)))

xlswrite('DNNresult.xls', hidden.net\_trainParam\_epochs, 'result', strcat('G',num2str(c)))

xlswrite('DNNresult.xls', acc\_train\_original \* 100, 'result', strcat('H',num2str(c)))

xlswrite('DNNresult.xls', acc\_test\_original \* 100, 'result', strcat('I',num2str(c)))

xlswrite('DNNresult.xls', acc\_train\_slope \* 100, 'result', strcat('J',num2str(c)))

xlswrite('DNNresult.xls', acc\_test\_slope \* 100, 'result', strcat('K',num2str(c)))

xlswrite('DNNresult.xls', acc\_train\_curvature \* 100, 'result', strcat('L',num2str(c)))

xlswrite('DNNresult.xls', acc\_test\_curvature \* 100, 'result', strcat('M',num2str(c)))

xlswrite('DNNresult.xls', accProb \* 100, 'result', strcat('N',num2str(c)))

xlswrite('DNNresult.xls', acc \* 100, 'result', strcat('O',num2str(c)))

xlswrite('DNNresult.xls', fix(T/3600)\*60+fix(mod(T,3600)/60), 'result', strcat('P',num2str(c)));

dt=fix(clock);

fprintf('时间记录：%d年 %d月 %d日 %d时 %d分 %d秒\n',dt(1),dt(2),dt(3),dt(4),dt(5),dt(6));

fprintf('程序运行总时间: %dh，%dmin, %ds\n',fix(T/3600),fix(mod(T,3600)/60),...

fix(mod(mod(T,3600),60)));

%% 在线预测

case 2

disp('在线预测！');

%加载网络参数

load para11

load para12

stackedAEOptTheta1=stackedAEOptTheta;

netconfig1=netconfig;

load para21

load para22

stackedAEOptTheta2=stackedAEOptTheta;

netconfig2=netconfig;

load para31

load para32

stackedAEOptTheta3=stackedAEOptTheta;

netconfig3=netconfig;

%预分配内存

pred2=zeros(1,length(Input\_Data.testLabels));

%原始数据所属类别

for i=1:length(Input\_Data.testLabels)

[pred\_test\_original,probability\_original] = stackedAEPredict(stackedAEOptTheta1, ...

inputdata\_options.sample\_num, hidden.num3, inputdata\_options.classes\_num,...

netconfig1, Input\_Data.testData\_original(:,i));

pred1\_2=pred\_test\_original;

%斜率数据所属类别

[pred\_test\_slope,probability\_slope] = stackedAEPredict(stackedAEOptTheta2, ...

inputdata\_options.sample\_num, hidden.num3, inputdata\_options.classes\_num,...

netconfig2, Input\_Data.testData\_slope(:,i));

pred2\_2=pred\_test\_slope;

%曲率数据所属类别

[pred\_test\_curvature,probability\_curvature] = stackedAEPredict(stackedAEOptTheta3, ...

inputdata\_options.sample\_num, hidden.num3, inputdata\_options.classes\_num,...

netconfig3, Input\_Data.testData\_curvature(:,i));

pred3\_2=pred\_test\_curvature;

%决策级分类融合策略

if pred1\_2==1&&pred2\_2==1&&pred3\_2==1

pred2(i)=1;

elseif pred1\_2==2&&pred2\_2==2&&pred3\_2==2

pred2(i)=2;

elseif pred1\_2==3&&pred2\_2==3&&pred3\_2==3

pred2(i)=3;

else

if (pred1\_2==1&&pred2\_2==1)||(pred1\_2==1&&pred3\_2==1)||...

(pred2\_2==1&&pred3\_2==1)

pred2(i)=1;

elseif (pred1\_2==2&&pred2\_2==2)||(pred1\_2==2&&pred3\_2==2)||...

(pred2\_2==2&&pred3\_2==2)

pred2(i)=2;

elseif (pred1\_2==3&&pred2\_2==3)||(pred1\_2==3&&pred3\_2==3)||...

(pred2\_2==3&&pred3\_2==3)

pred2(i)=3;

else

if i>9

s=zeros(1,10);

sum1=0;

sum2=0;

sum3=0;

for j=1:10

s(j)=pred2(i-j);

if s(j)==1

sum1=sum1+1;

elseif s(j)==2

sum2=sum2+1;

else

sum3=sum3+1;

end

end

A=[sum1,sum2,sum3];

maxRes=max(A);

if maxRes==1

pred2(i)=1;

elseif maxRes==2

pred2(i)=2;

else

pred2(i)=3;

end

else

pred2(i)=1;

end

end

end

fprintf('第%d个数据的故障类别: %d\n', i,pred2(i));

pause(1);

end

otherwise

disp('输入错误！');

end

fprintf('运行完毕');

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%% DNN子程序 %%

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%5%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function dnn(Input\_Data,inputdata\_options,hidden)

%% 参数

fprintf('原始数据训练测试中......');

trainData=Input\_Data.trainData\_original;

trainLabels=Input\_Data.trainLabels;

testData=Input\_Data.testData\_original;

testLabels=Input\_Data.testLabels;

sample\_num=inputdata\_options.sample\_num;

classes\_num=inputdata\_options.classes\_num;

%% STEP 1 原始数据通过堆叠自动编码器进行特征抽取

[options]=Creat\_AE\_Mode(inputdata\_options,hidden,trainData);

%% STEP 2 softmax模型训练过程%%%%%%%%%%%%%%%%%%

sae3Features=options.feature3;%第三层的特征

softlambda = 1e-2; %权值下降参数 每一次迭代更新参数时用到

softoptions.maxIter = 100;%最大迭代次数

softmaxModel = softmaxTrain(hidden.num3, classes\_num, softlambda, ...

sae3Features, trainLabels, softoptions);

saeSoftmaxOptTheta = softmaxModel.optTheta(:);

%% STEP 3 fine -tune微调参数 %%%%%%%%%%%%%%%%%%%%%%%

% 将自动编码器学习到的参数堆叠起来

stack = cell(3,1);%3x1的元胞数组

stack{1}.w=options.w1;

stack{2}.w=options.w2;

stack{3}.w=options.w3;

stack{1}.b=options.b1;

stack{2}.b=options.b2;

stack{3}.b=options.b3;

[stackparams, netconfig] = stack2params(stack);

stackedAETheta = [ saeSoftmaxOptTheta ; stackparams ];

lambda = 1e-3; % 权值下降参数

options.Method = 'lbfgs'; %拟牛顿限制内存法

options.alpha=0.05; %学习率

options.maxIter =hidden.net\_trainParam\_epochs; %最大迭代次数

%% STEP 4 最小代价函数minFunc

[stackedAEOptTheta, ~] = minFunc(@(p)stackedAECost(p,sample\_num,hidden.num3,...

classes\_num, netconfig,lambda,trainData,trainLabels),...

stackedAETheta,options);

%% STEP 5: 输出原始训练数据分类精度

[pred\_train\_original,~] = stackedAEPredict(stackedAEOptTheta, sample\_num, hidden.num3, ...

classes\_num, netconfig, trainData);

acc\_train\_original = mean(trainLabels(:) == pred\_train\_original(:));

fprintf('训练原始数据分类正确率: %0.2f%%\n', acc\_train\_original \* 100);

%% STEP 6: 输出原始测试数据分类精度

[pred\_test\_original,probability\_original] = stackedAEPredict(stackedAEOptTheta, sample\_num, hidden.num3, ...

classes\_num, netconfig, testData);

acc\_test\_original = mean(trainLabels(:) == pred\_test\_original(:));

fprintf('测试原始数据分类正确率: %0.2f%%\n', acc\_test\_original \* 100);

%% 保存原始数据训练好的参数

save para11 stackedAEOptTheta

save para12 netconfig

%% 斜率数据%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

fprintf('斜率数据训练测试中......');

%% STEP 1 原始数据通过堆叠自动编码器进行特征抽取

trainData=Input\_Data.trainData\_slope;

testData=Input\_Data.testData\_slope;

[options]=Creat\_AE\_Mode(inputdata\_options,hidden,trainData);

%% STEP 2 softmax模型训练过程%%%%%%%%%%%%%%%%%%

sae3Features=options.feature3;%第三层的特征

softlambda = 1e-3; %权值下降参数 每一次迭代更新参数时用到

softoptions.maxIter = 100;%最大迭代次数

softmaxModel = softmaxTrain(hidden.num3, classes\_num, softlambda, ...

sae3Features, trainLabels, softoptions);

saeSoftmaxOptTheta = softmaxModel.optTheta(:);

%% STEP 3 fine -tune微调参数 %%%%%%%%%%%%%%%%%%%%%%%

% 将自动编码器学习到的参数堆叠起来

stack = cell(3,1);%3x1的元胞数组

stack{1}.w=options.w1;

stack{2}.w=options.w2;

stack{3}.w=options.w3;

stack{1}.b=options.b1;

stack{2}.b=options.b2;

stack{3}.b=options.b3;

[stackparams, netconfig] = stack2params(stack);

stackedAETheta = [ saeSoftmaxOptTheta ; stackparams ];

lambda = 1e-3; % 权值下降参数

options.Method = 'lbfgs'; %拟牛顿限制内存法

options.alpha=0.05; %学习率

options.maxIter =hidden.net\_trainParam\_epochs; %最大迭代次数

%% STEP 4 最小代价函数minFunc

[stackedAEOptTheta, ~] = minFunc(@(p)stackedAECost(p,sample\_num,hidden.num3,...

classes\_num, netconfig,lambda,trainData,trainLabels),...

stackedAETheta,options);

%% STEP 5: 输出原始训练数据分类精度

[pred\_train\_slope,~] = stackedAEPredict(stackedAEOptTheta, sample\_num, hidden.num3, ...

classes\_num, netconfig, trainData);

acc\_train\_slope = mean(trainLabels(:) == pred\_train\_slope(:));

fprintf('训练斜率数据分类正确率: %0.2f%%\n', acc\_train\_slope \* 100);

%% STEP 6: 输出斜率测试数据分类精度

[pred\_test\_slope,probability\_slope] = stackedAEPredict(stackedAEOptTheta, sample\_num,...

hidden.num3,classes\_num, netconfig, testData);

acc\_test\_slope = mean(testLabels(:) == pred\_test\_slope(:));

fprintf('测试斜率数据分类正确率: %0.2f%%\n', acc\_test\_slope \* 100);

%% 保存斜率数据训练好的参数

save para21 stackedAEOptTheta

save para22 netconfig

%% 曲率数据%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

fprintf('曲率数据训练测试中......');

%% STEP 1 原始数据通过堆叠自动编码器进行特征抽取

trainData=Input\_Data.trainData\_curvature;

testData=Input\_Data.testData\_curvature;

[options]=Creat\_AE\_Mode(inputdata\_options,hidden,trainData);

%% STEP 2 softmax模型训练过程%%%%%%%%%%%%%%%%%%

sae3Features=options.feature3;%第三层的特征

softlambda = 1e-3; %权值下降参数 每一次迭代更新参数时用到

softoptions.maxIter = 100;%最大迭代次数

softmaxModel = softmaxTrain(hidden.num3, classes\_num, softlambda, ...

sae3Features, trainLabels, softoptions);

saeSoftmaxOptTheta = softmaxModel.optTheta(:);

%% STEP 3 fine -tune微调参数 %%%%%%%%%%%%%%%%%%%%%%%

% 将自动编码器学习到的参数堆叠起来

stack = cell(3,1);%3x1的元胞数组

stack{1}.w=options.w1;

stack{2}.w=options.w2;

stack{3}.w=options.w3;

stack{1}.b=options.b1;

stack{2}.b=options.b2;

stack{3}.b=options.b3;

[stackparams, netconfig] = stack2params(stack);

stackedAETheta = [ saeSoftmaxOptTheta ; stackparams ];

lambda = 1e-3; % 权值下降参数

options.Method = 'lbfgs'; %拟牛顿限制内存法

options.alpha=0.05; %学习率

options.maxIter =hidden.net\_trainParam\_epochs; %最大迭代次数

%% STEP 4 最小代价函数minFunc

[stackedAEOptTheta, ~] = minFunc(@(p)stackedAECost(p,sample\_num,hidden.num3,...

classes\_num, netconfig,lambda,trainData,trainLabels),...

stackedAETheta,options);

%% STEP 5: 输出原始训练数据分类精度

[pred\_train\_curvature,~] = stackedAEPredict(stackedAEOptTheta, sample\_num, hidden.num3, ...

classes\_num, netconfig, trainData);

acc\_train\_curvature = mean(trainLabels(:) == pred\_train\_curvature(:));

fprintf('训练曲率数据分类正确率: %0.2f%%\n', acc\_train\_curvature \* 100);

%% STEP 6: 输出测试曲率数据分类精度

[pred\_test\_curvature,probability\_curvature] = stackedAEPredict(stackedAEOptTheta, sample\_num,...

hidden.num3,classes\_num, netconfig, testData);

acc\_test\_curvature = mean(testLabels(:) == pred\_test\_curvature(:));

fprintf('测试曲率数据分类正确率: %0.2f%%\n', acc\_test\_curvature \* 100);

%% 保存曲率数据训练好的参数

save para31 stackedAEOptTheta

save para32 netconfig

%% 保存数据

save ('classesResult.mat','acc\_train\_original','acc\_test\_original','acc\_train\_slope','acc\_test\_slope',...

'acc\_train\_curvature','acc\_test\_curvature');

save ('probabilitypred.mat','probability\_original','probability\_slope','probability\_curvature',...

'pred\_test\_original','pred\_test\_slope','pred\_test\_curvature','testLabels');

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%% Creat\_AE\_Mode子程序 %%

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%5%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function[options]=Creat\_AE\_Mode(inputdata\_options,hidden,trainData)

%% 自动编码器简介

%自动编码器是一种无监督的学习算法，利用反向传播算法（bp算法），让输出值等于输入值

%随着隐层的增加，每层神经元个数的递减，自动编码器可以学习到数据的一些压缩表示。

%比如第一隐层用600维表示1200维的数据，到第三隐层仅用一百维就可以表示1200维的数据

%这样数据之间的相关性就必然增加了很多，所以说自动编码器就是学习输入数据相关性的一种表示方法。

%@衡量网络性能所用的函数。默认的performFcn是mse，而可选的也只有四种，

%1、mae Mean absolute error 2、mse Mean squared error

%3、msereg Mean squared error w/reg 4、还有sse

%@权值学习算法，可以有：1、带动量的梯度下降法（traingdm）,

%2、L-M优化算法（trainlm）,3、量化共轭梯度法（traingdm）

%除了上面三种还可以有：traindx,trainda等。

%@节点传递函数（神经元激活函数），可以有：1、双曲正切函数（tansig）

%2、单极性s型函数（logsig）3、线性函数（purelin）

%% 取出参数

learn\_rare=hidden.learn\_rare;

sample\_total=inputdata\_options.sample\_total;

sample\_num=inputdata\_options.sample\_num;

classes\_num=inputdata\_options.classes\_num;

n\_columns=inputdata\_options.n\_columns;

%% STEP1 建立第一层自动编码器 %%%%%%%%%%%%%%%%%%%%%

net1=feedforwardnet(hidden.num1);

net1.name = 'Autoencoder';

net1.layers{1}.name = 'Encoder';

net1.layers{2}.name = 'Decoder';

net1.layers{1}.initFcn = 'initwb';

net1.layers{2}.initFcn = 'initwb';

net1.inputWeights{1,1}.initFcn = 'rands';

net1.inputWeights{2,1}.initFcn = 'rands';

net1.biases{1}.initFcn='initzero';

net1.biases{2}.initFcn='initzero';

net1.initFcn = 'initlay';

net1.performFcn='mse';

net1.trainFcn='traingdm';

net1.layers{1}.transferFcn='tansig';

net1.layers{2}.transferFcn='tansig';

net1.trainParam.epochs=hidden.net\_trainParam\_epochs; %训练次数

net1.trainParam.mc=0.05; %动量

net1.trainParam.lr=learn\_rare;%学习率

net1=init(net1); %初始化网络

[net1]=train(net1,trainData,trainData); %训练网络

%% 提取第一隐层特征%%%%%%%%%%

%获取编码网络的权值和偏置

options.w1=net1.iw{1,1}; % 只要编码网络的权值即可，解码网络只是在训练的时候有用

options.b1=net1.b{1}; % 训练后的编码网络的偏置等于（hidden\_num1\*1）

a1=options.w1\*trainData+options.b1\*ones(1,n\_columns\*classes\_num);

feature1=tansig(a1); %用双曲正切激活函数得到第一隐层的特征

%% STEP2建立第二层自编码器 %%%%%%%%%%%%%%

net2=feedforwardnet(hidden.num2);

net2.name = 'Autoencoder';

net2.layers{1}.name = 'Encoder';

net2.layers{2}.name = 'Decoder';

net2.layers{1}.initFcn = 'initwb';

net2.layers{2}.initFcn = 'initwb';

net2.inputWeights{1,1}.initFcn = 'rands';

net2.inputWeights{2,1}.initFcn = 'rands';

net2.biases{1}.initFcn='initzero';

net2.biases{2}.initFcn='initzero';

net2.initFcn = 'initlay';

net2.performFcn='mse';

net2.trainFcn='traingdm';

net2.layers{1}.transferFcn='tansig';

net2.layers{2}.transferFcn='tansig';

net2.trainParam.epochs=hidden.net\_trainParam\_epochs;

net2.trainParam.mc=0.05;

net2.trainParam.lr=learn\_rare;

net2=init(net2);

net2=train(net2,feature1,feature1);

%% 提取第二隐层特征%%%%%%%%%%

options.w2=net2.iw{1,1};

options.b2=net2.b{1};

a2=options.w2\*feature1+options.b2\*ones(1,n\_columns\*classes\_num);

feature2=tansig(a2);

%% STEP 3建立第三个自编码器%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

net3=feedforwardnet(hidden.num3);%建立第三层自编码器，神经元个数hidden.num3

net3.name = 'Autoencoder';

net3.layers{1}.name = 'Encoder';

net3.layers{2}.name = 'Decoder';

net3.layers{1}.initFcn = 'initwb';

net3.layers{2}.initFcn = 'initwb';

net3.inputWeights{1,1}.initFcn = 'rands';

net3.inputWeights{2,1}.initFcn = 'rands';

net3.biases{1}.initFcn='initzero';

net3.biases{2}.initFcn='initzero' ;

net3.initFcn = 'initlay';

net3.performFcn='mse';

net3.trainFcn='traingdm';

net3.layers{1}.transferFcn='tansig';

net3.layers{2}.transferFcn='tansig';

net3.trainParam.epochs=hidden.net\_trainParam\_epochs;

net3.trainParam.mc=0.05;

net3.trainParam.lr=learn\_rare;

net3=init(net3);

net3=train(net3,feature2,feature2);

%% 提取第三隐层特征%%%%%%%%%%

options.w3=net3.iw{1,1};

options.b3=net3.b{1};

a3=options.w3\*feature2+options.b3\*ones(1,n\_columns\*classes\_num);

options.feature3=tansig(a3);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%% Add\_Original\_Data()子程序 %%%

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function [Input\_Data,testdata]=Add\_Original\_Data(inputdata\_options) %

%% 添加相对路径，就算换电脑或者换文件所在目录都能直接打开程序并运行

s=what; % 获得当前目录

s.path % 当前路径

filep=fullfile(s.path,'data') ;% 需要加载的数据所在文件夹的路径

addpath(filep); % 添加路径

%% 加载原始数据：任意选取classes\_num种数据

load Normal\_1

load IR007\_0

load B021\_3

%% 原始数据

sample\_total=inputdata\_options.sample\_total;%训练数据总长

sample\_num=inputdata\_options.sample\_num;%训练数据维度

n\_columns=inputdata\_options.n\_columns;%数据维度

%% 取出原始数据

x1=(X098\_DE\_time(1:sample\_total));%Normal\_1数据1(1\*sample\_num)

x2=(X109\_DE\_time(1:sample\_total));%IR007\_0数据2(1\*sample\_num)

x3=(X229\_DE\_time(1:sample\_total));%B007\_0数据3(1\*sample\_num)

%% 为斜率&&曲率数据预分配内存

x1\_slope=zeros(1,sample\_total);

x1\_curvature=zeros(1,sample\_total);

x2\_slope=zeros(1,sample\_total);

x2\_curvature=zeros(1,sample\_total);

x3\_slope=zeros(1,sample\_total);

x3\_curvature=zeros(1,sample\_total);

%% 求斜率&&曲率

for i=1:sample\_total-1

%% 斜率

x1\_slope(i)=(x1(i+1)-x1(i));

x2\_slope(i)=(x2(i+1)-x2(i));

x3\_slope(i)=(x3(i+1)-x3(i));

%% 曲率

x1\_curvature(i)=(x1\_slope(i+1)-x1\_slope(i));

x2\_curvature(i)=(x2\_slope(i+1)-x2\_slope(i));

x3\_curvature(i)=(x3\_slope(i+1)-x3\_slope(i));

end

%% 给斜率曲率的最后一个值通过简单赋值处理，以后可以拟合这个数据或者求取后将最后一位输入数据舍弃

x1\_slope(sample\_total)=x1\_slope(sample\_total-1);

x2\_slope(sample\_total)=x2\_slope(sample\_total-1);

x3\_slope(sample\_total)=x3\_slope(sample\_total-1);

x1\_curvature(sample\_total)=x1\_curvature(sample\_total-1);

x2\_curvature(sample\_total)=x2\_curvature(sample\_total-1);

x3\_curvature(sample\_total)=x3\_curvature(sample\_total-1);

%% 按时间划分

bb11=zeros(sample\_num,n\_columns);

bb12=zeros(sample\_num,n\_columns);

bb13=zeros(sample\_num,n\_columns);

aa11=zeros(sample\_num,n\_columns);

aa12=zeros(sample\_num,n\_columns);

aa13=zeros(sample\_num,n\_columns);

cc11=zeros(sample\_num,n\_columns);

cc12=zeros(sample\_num,n\_columns);

cc13=zeros(sample\_num,n\_columns);

b11=zeros(sample\_num,n\_columns);

b12=zeros(sample\_num,n\_columns);

b13=zeros(sample\_num,n\_columns);

a11=zeros(sample\_num,n\_columns);

a12=zeros(sample\_num,n\_columns);

a13=zeros(sample\_num,n\_columns);

c11=zeros(sample\_num,n\_columns);

c12=zeros(sample\_num,n\_columns);

c13=zeros(sample\_num,n\_columns);

for i=1:sample\_num

aa11(i,:)=x1(i:n\_columns+i-1);

aa12(i,:)=x2(i:n\_columns+i-1);

aa13(i,:)=x3(i:n\_columns+i-1);

bb11(i,:)=x1\_slope(i:n\_columns+i-1);

bb12(i,:)=x2\_slope(i:n\_columns+i-1);

bb13(i,:)=x3\_slope(i:n\_columns+i-1);

cc11(i,:)=x1\_curvature(i:n\_columns+i-1);

cc12(i,:)=x2\_curvature(i:n\_columns+i-1);

cc13(i,:)=x3\_curvature(i:n\_columns+i-1);

a11(i,:)=x1(n\_columns+sample\_num-1+i:n\_columns+sample\_num-1+n\_columns+i-1);

a12(i,:)=x2(n\_columns+sample\_num-1+i:n\_columns+sample\_num-1+n\_columns+i-1);

a13(i,:)=x3(n\_columns+sample\_num-1+i:n\_columns+sample\_num-1+n\_columns+i-1);

b11(i,:)=x1\_slope(n\_columns+sample\_num-1+i:n\_columns+sample\_num-1+n\_columns+i-1);

b12(i,:)=x2\_slope(n\_columns+sample\_num-1+i:n\_columns+sample\_num-1+n\_columns+i-1);

b13(i,:)=x3\_slope(n\_columns+sample\_num-1+i:n\_columns+sample\_num-1+n\_columns+i-1);

c11(i,:)=x1\_curvature(n\_columns+sample\_num-1+i:n\_columns+sample\_num-1+n\_columns+i-1);

c12(i,:)=x2\_curvature(n\_columns+sample\_num-1+i:n\_columns+sample\_num-1+n\_columns+i-1);

c13(i,:)=x3\_curvature(n\_columns+sample\_num-1+i:n\_columns+sample\_num-1+n\_columns+i-1);

end

Input\_Data.trainData\_original=[aa11,aa12,aa13];

Input\_Data.trainData\_slope=[bb11,bb12,bb13];

Input\_Data.trainData\_curvature=[cc11,cc12,cc13];

Input\_Data.testData\_original=[a11,a12,a13];

Input\_Data.testData\_slope=[b11,b12,b13];

Input\_Data.testData\_curvature=[c11,c12,c13];

%% 标签

Input\_Data.trainLabels=[ones(1,n\_columns),repmat(2,1,n\_columns),repmat(3,1,n\_columns)];

Input\_Data.testLabels=[ones(1,n\_columns),repmat(2,1,n\_columns),repmat(3,1,n\_columns)];

% long=1000;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%% softmaxTrain()子程序%%%

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%% softmaxTrain Train%%%%%%%%%%%%%%

%输入参数： inputSize: 输入向量大小（自动编码器第三隐层神经元个数）

% numClasses: 分类数（10）

% lambda: 权重下降参数1e-3; %权值下降参数 每一次迭代更新参数时用到

% inputData: 输入数据（第三层特征）

% labels: 输入训练数据 的 标签

% options (optional): options

% options.maxIter: 训练时的迭代次数

%输入参数：softmaxModel（结构体）

function [softmaxModel] = softmaxTrain(inputSize, numClasses, lambda, inputData, labels, options)

%% exist('im', 'var')是检测im中的变量是否存在，如果不存在返回0，存在返回1。

%~exist('im', 'var')是对结果取非运算

if ~exist('options', 'var')%如果不存在一个options的结构体则执行if里面的句子

options = struct;%定义一个结构体options（由于之前定义了所以这个语句未执行）

end

%% 通过函数fieldnames来获取字段名称（结构体的所有属性）

%通过函数isfield来判断是否存在某一字段

%如果结构体options中存在maxIter属性，则为1，再~取反，则为0，那么if里面的句子不执行。

if ~isfield(options, 'maxIter')

options.maxIter = 200;%实际未执行

end

%% 初始化参数%%%%%%%%%

theta = 0.005 \* randn(numClasses \* inputSize, 1);%随机初始化参数theta（1000x1）

%% 用最小化代价函数minFunc去训练参数

addpath minFunc/ %添加最小化函数所在的文件夹的路径

%拟牛顿限制内存法%比梯度下降法好的多

%给结构体添加属性可以直接写，但是删除要用Rmfield函数

options.Method = 'lbfgs';

%调用minFunc函数去调整参数Theta。函数返回的softmaxOptTheta：1000x1

[softmaxOptTheta] = minFunc( @(p) softmaxCost(p,numClasses, inputSize, lambda,inputData, labels),theta, options);

% 调整下Theta的结构为10x100

softmaxModel.optTheta = reshape(softmaxOptTheta, numClasses, inputSize);

softmaxModel.inputSize = inputSize;%100

softmaxModel.numClasses = numClasses;%10

end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%% softmaxCost()子程序 %%%

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function [cost, grad] = softmaxCost(theta, numClasses, inputSize, lambda, data, labels)

% Unroll the parameters from theta

theta = reshape(theta, numClasses, inputSize);

numCases = size(data, 2);

groundTruth = full(sparse(labels, 1:numCases, 1));

cost = 0;

thetagrad = zeros(numClasses, inputSize);

%% ---------- YOUR CODE HERE --------------------------------------

% Instructions: Compute the cost and gradient for softmax regression.

% You need to compute thetagrad and cost.

% The groundTruth matrix might come in handy.

M = bsxfun(@minus, theta\*data,max(theta\*data,[],1));

M = exp(M);

p = bsxfun(@rdivide,M,sum(M,1));

cost = -1/numCases \* groundTruth(:)' \* log(p(:)) + lambda /2 \* theta(:)'\* theta(:);

thetagrad = -1/numCases \* (groundTruth - p)\*data' + lambda \* theta;

% ------------------------------------------------------------------

% Unroll the gradient matrices into a vector for minFunc

grad = [thetagrad(:)];

end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%% params2stack ()子程序 %%

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function stack = params2stack(params, netconfig)

% Map the params (a vector into a stack of weights)

depth = numel(netconfig.layersizes);

stack = cell(depth,1);

prevLayerSize = netconfig.inputsize; % the size of the previous layer

curPos = double(1); % mark current position in parameter vector

for d = 1:depth

% Create layer d

stack{d} = struct;

% Extract weights

wlen = double(netconfig.layersizes{d} \* prevLayerSize);

stack{d}.w = reshape(params(curPos:curPos+wlen-1), netconfig.layersizes{d}, prevLayerSize);

curPos = curPos+wlen;

% Extract bias

blen = double(netconfig.layersizes{d});

stack{d}.b = reshape(params(curPos:curPos+blen-1), netconfig.layersizes{d}, 1);

curPos = curPos+blen;

% Set previous layer size

prevLayerSize = netconfig.layersizes{d};

end

end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%% stack2params()子程序 %%

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function [params, netconfig] = stack2params(stack)

% Setup the compressed param vector

params = [];

for d = 1:numel(stack)

% This can be optimized. But since our stacks are relatively short, it

% is okay

params = [params ; stack{d}.w(:) ; stack{d}.b(:) ];

% Check that stack is of the correct form

assert(size(stack{d}.w, 1) == size(stack{d}.b, 1), ...

['The bias should be a \*column\* vector of ' ...

int2str(size(stack{d}.w, 1)) 'x1']);

if d < numel(stack)

assert(size(stack{d}.w, 1) == size(stack{d+1}.w, 2), ...

['The adjacent layers L' int2str(d) ' and L' int2str(d+1) ...

' should have matching sizes.']);

end

end

if nargout > 1

% Setup netconfig

if numel(stack) == 0

netconfig.inputsize = 0;

netconfig.layersizes = {};

else

netconfig.inputsize = size(stack{1}.w, 2);

netconfig.layersizes = {};

for d = 1:numel(stack)

netconfig.layersizes = [netconfig.layersizes ; size(stack{d}.w,1)];

end

end

end

end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%% stackedAECost子程序 %%

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function [ cost, grad ] = stackedAECost(theta, inputSize, hiddenSize, ...

numClasses, netconfig, ...

lambda, data, labels)

softmaxTheta = reshape(theta(1:hiddenSize\*numClasses), numClasses, hiddenSize);

% Extract out the "stack"

stack = params2stack(theta(hiddenSize\*numClasses+1:end), netconfig)

% You will need to compute the following gradients

softmaxThetaGrad = zeros(size(softmaxTheta));

stackgrad = cell(size(stack));

for d = 1:numel(stack)

stackgrad{d}.w = zeros(size(stack{d}.w));

stackgrad{d}.b = zeros(size(stack{d}.b));

end

cost = 0; % You need to compute this

% You might find these variables useful

M = size(data, 2);

groundTruth = full(sparse(labels, 1:M, 1));

depth = numel(stack);

z = cell(depth+1,1);

a = cell(depth+1,1);

a{1} = data;

for layer = 1:depth

% size(stack{layer}.w )

% size(a{layer})

% size(stack{layer}.w \* a{layer})

% size(stack{layer}.b)

%z{depth+1} = stack{layer}.w \* a{layer} + repmat(stack{layer}.b, 1, size(a{layer},2));

z{layer+1} = bsxfun(@plus,stack{layer}.w \* a{layer}, stack{layer}.b);

a{layer+1} = sigmoid(z{layer+1});

end

P = softmaxTheta\* a{depth+1};

P = bsxfun(@minus, P ,max(P,[],1));

P = exp(P);

p = bsxfun(@rdivide,P,sum(P,1));

cost = -1/M \* groundTruth(:)' \* log(p(:)) + lambda /2 \* softmaxTheta(:)'\* softmaxTheta(:);

softmaxThetaGrad = -1/M \* (groundTruth - p)\*a{depth+1}' + lambda \* softmaxTheta;

delta = cell(depth+1);

delta{depth+1} = -(softmaxTheta' \* (groundTruth - p)) .\* a{depth+1} .\* (1 - a{depth+1});

for layer = (depth:-1:2)

delta{layer} = (stack{layer}.w' \* delta{layer+1}) .\* a{layer} .\* (1-a{layer});

end

for layer = (depth:-1:1)

stackgrad{layer}.w =(1/M) \* delta{layer+1} \* a{layer}';

stackgrad{layer}.b = (1/M) \* sum(delta{layer+1}, 2);

end

%% Roll gradient vector

grad = [softmaxThetaGrad(:) ; stack2params(stackgrad)];

end

% You might find this useful

function sigm = sigmoid(x)

sigm = 1 ./ (1 + exp(-x));

end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%% stackedAECost子程序 %%

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function [pred,probability] = stackedAEPredict(theta, ~, hiddenSize, numClasses, netconfig, data)

softmaxTheta = reshape(theta(1:hiddenSize\*numClasses), numClasses, hiddenSize);

% Extract out the "stack"

stack = params2stack(theta(hiddenSize\*numClasses+1:end), netconfig);

%% ---------- YOUR CODE HERE --------------------------------------

% Instructions: Compute pred using theta assuming that the labels start

% from 1.

depth = numel(stack);

z = cell(depth+1,1);

a = cell(depth+1, 1);

a{1} = data;

for layer = (1:depth)

% z{layer+1} = stack{layer}.w \* a{layer} + repmat(stack{layer}.b, [1, size(a{layer},2)]);

z{layer+1} = bsxfun(@plus,stack{layer}.w \* a{layer}, stack{layer}.b);

a{layer+1} = sigmoid(z{layer+1});

end

probability=softmaxTheta \* a{depth+1};

[~, pred] = max(probability);

end

% You might find this useful

function sigm = sigmoid(x)

sigm = 1 ./ (1 + exp(-x));

end